REMARKS/ARGUMENTS

Claims 17-23 and 35 are pending in the application. Claims 17-23 and 35 are rejected under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claims 1-11 of U.S. Patent No. 6,792,446. Claims 17-23 and 35 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gulati et al. (Performance Study of a Concurrent Multithreaded Processor) in view of Loikkanen et al. (A Fine-Grain Multithreading Superscalar Architecture) further in view of Steely, Jr. et al. (U.S. Patent No. 5,197,132). With regard to the obviousness type double patenting rejections, please see previously submitted terminal disclaimer dated May 20, 2005. Claim 17 is amended to put it into better form. New claims 37-40 are hereby added.

Applicants respectfully submit the cited references do not teach, suggest or disclose "[a] method comprising: ... determining that a first thread has stalled; temporarily storing one or more instructions of the first thread in a replay queue..." (e.g., as described in claim 17).

The Office Action asserts Gulati teaches determining that a first thread is stalled at page 297, column 1. *See* Office Action, page 3, paragraph 7. Applicants disagree. The cited column 1 of page 297 states:

> Upon detecting certain instructions, the decoder sends a switch signal to the fetch mechanism. The fetch mechanism ceases fetching for the currently active thread and switches to another one. The instructions that can trigger a context switch are:
>
> integer divide
> floating point multiply or divide
> a synchronization primitive
> a long-latency I/O operation
>
> Cache misses do not belong to this list, because the decision to switch is being made at the decode stage.

> Figures 3 and 4 show the cycles of execution for Group I and II benchmarks respectively, with the three different fetch policies. The base case is also shown for sake of comparison. Each benchmark is compiled to run with four parallel threads, which is the default number, as per Table 2. All other hardware features correspond to the configuration in the same table. "LL#n" refers to Livermore loop #n. Performance-wise, True RR and Masked RR emerge as about equivalent . While Masked RR has distinct advantages, it has the drawback of sometimes masking threads out unnecessarily. Threads may get masked owing to short-latency operations, and if this occurs frequently, it would result in a sparsely occupied SU. Conditional switch, which has been included for sake of comparison, has similar performance. This implies that the latencies of operations that trigger a context switch for this policy are not a bottleneck in the processor's execution rate. Of the three policies, True Round Robin is the easiest to implement.
>
> Figures 5 and 6 present the results of execution of the benchmarks with 1, 2, 3, 4, 5, and 6 threads. We shall use the term "peak improvement" of a benchmark to refer to its maximum improvement among all multithreaded simulations, i.e. the maximum observed value of speedup among 2, 3, 4, 5, or 6 threads.

Applicants submit the cited section is not directed toward stalled instructions at all, but rather the fetching of instructions, and more specifically, the criteria upon which a "switch" is implemented during fetching. For example, the first paragraph of the cited section describes the introduction of a "switch" instruction. At the execution of the "switch" instruction, execution of the active thread is terminated and "switched" with another thread. The criteria for this determination is described, including an integer divide or a floating point.

First, Applicants assert that the cited section does not address "stalls" at all anywhere. Furthermore, Applicants submit this section is not directed toward resource conflicts, execution-dependency issues, or branch progression hazards or any other issues traditionally associated with stalls. In fact, the cited reference specifically states that cache misses (possibly due to dependency issues) *do not* belong on the list of reasons for the "switch" to occur. Applicants submit that this "switch" mechanism described in the Gulati implemented upon floating point

type scenarios is inadequate to address stalled threads as described in the embodiment of claim 17.

The second to last paragraph of the cited section describes Figures 3 and 4 of the Gulati reference, including a comparison the Livermore, Laplace, Matrix, and Water fetch policies. It also compares the True Round Robin (RR) to the Masked RR, and determines that the performance under the two are similar. Applicants note that this paragraph does not describe or pertain to stalls at all.

The last paragraph discusses the application of the references to multithreaded simulations. Applicants note that this paragraph does not describe or pertain to stalls at all.

Therefore, since the Gulati reference does not teach at least "*...determining that a first thread has stalled*; temporarily storing one or more instructions of the first thread in a replay queue...", the cited reference fails to adequately support a proper 35 U.S.C. §103(a) rejection of independent claim 17. Independent claims 35 contains substantively similar limitations.

Applicants further submit the cited reference fails to teach, suggest or describe "determining that a first thread has stalled; *temporarily storing one or more instructions of the first thread* in a replay queue..." (e.g., as described in claim 17).

The Office Action asserts that Loikkanen teaches temporarily storing one or more instructions of the first thread in a queue, citing page 166, column 1, section "Remote Loads and Stores". *See* page 3, paragraph 9. The cited section states:

> Remote Loads and Stores. Remote loads are issued into a Load Queue of TSIBs of the Load Unit. *Remote stores are buffered in the Store Queue in the same manner as local stores (to assure in-order completion). When a remote load instruction is issued to the Load Unit, the instruction is added to the Load Queue and the request is sent out to the network.* Completion of the remote load instruction occurs when the requested data

> arrives. (The thread was suspended when the instruction was decoded.) *(emphasis added)*

Applicants submit the cited section does not describe stalled instructions at all. Instead, it describes buffering "remote stores" in a Store queue in the same manner as "local stores". Therefore, the cited section is merely describing buffering "remote" data in the same manner as "local" data to ensure continuity. No description of stalled conditions or criteria is included in the cited sections. The section continues to describe the loading of this "remote" data into a "load unit" and the "load queue" before sending it over the network.

Applicants submit that since the cited section does not pertain to stalled instructions at all (for at least the reasons detailed above), the cited section is inadequate to "*determining that a first thread has stalled*; temporarily storing one or more instructions *of the first thread* in a replay queue..." (e.g., as described in claim 1). Independent claim 35 contains similar allowable limitations.

Steely fails to make up for the multiple deficiencies of Gulati. Steely is directed toward a register mapping system used in the execution of instructions processed through a computer pipeline. Steely is at least "determining that a first thread has stalled; temporarily storing one or more instructions of the first thread in a replay queue..." (e.g., as described in claim 17).

Loikkenen fails for similar reasons as well. Although Loikkenen is directed toward a fine grain multithreading superscaler architecture, it does not describe at least "determining that a first thread has stalled; temporarily storing one or more instructions of the first thread in a replay queue..." (e.g., as described in claim 17).

Since each and every limitation is not found in the cited references, the cited references cannot be combined to adequately form the basis of a proper 35 U.S.C. §103(a) rejection of

independent claim 17. Independent claim 35 contains substantively similar limitations and therefore is allowable for similar reasons. Claims 18-23 and 37-40 depend from allowable independent claims 17 and 35 and therefore are in condition for allowance as well. Furthermore, Applicants assert that the two references cannot be combined without the use of impermissible hindsight reasoning.

For at least all the above reasons, the Applicants respectfully submit that this application is in condition for allowance. A Notice of Allowance is earnestly solicited.

The Examiner is invited to contact the undersigned at (408) 975-7500 to discuss any matter concerning this application. The Office is hereby authorized to charge any additional fees or credit any overpayments under 37 C.F.R. § 1.16 or § 1.17 to Deposit Account No. **11-0600**.

Respectfully submitted,
KENYON & KENYON LLP

Dated: March 23, 2006 By: _____
Sumit Bhattacharya
(Reg. No. 51,469)
Attorneys for Intel Corporation

KENYON & KENYON LLP
333 W. San Carlos St., Suite 600
San Jose, CA 95110
Telephone: (408) 975-7500
Facsimile: (408) 975-7501

Customer No. 25693